CS305 – Programming Languages
Fall 2024-2025

HOMEWORK 3

Implementing a Semantic Analyzer for LCD

Due date: December 13, 2023 @ 23:55

NOTE

Only SUCourse submission is allowed. No submission by e-mail. Please see the note at the end of this document for late submission policy.

1 Introduction

In this homework, you will implement a tool that includes a simple semantic analyzer for LCD language. Detailed information about this programming language was given in the second homework document. You can check it for more information.

The tool that you will implement in this homework will first check if a given LCD program has any syntax errors grammatically. The tool will perform semantic checks if there are no syntax errors. If these checks are passed, then your tool will print out the evaluation results. Read the rest of the document for more information.

2 Parser and Scanner

The scanner and the parser, which you can use to implement this homework, will be provided to you. The semantic analysis will require you to implement an attribute grammar. You can start from the scanner/parser files provided, or of course, you can write your own versions of scanner and parser from scratch.

3 Semantic Rules

Your semantic analyzer should start by performing an analysis for the following semantic rules. Your semantic analyzer must print out an error message for each violation of these rules. Note that after printing out an error message, your semantic analyzer must not terminate and keep working to find further violations, if any exist.

Rule 1 (Undeclared Identifiers): *Every identifier that is used in either the circuit design block or the evaluations block must be declared in the declarations block.*

Identifiers are used in the circuit design or the evaluations blocks. All such identifiers must be declared in the declarations block. Note that this includes the identifiers used on the left-hand side and the identifiers used in the expressions on the right-hand side of the assignments. If an error occurs, the following error message will be printed:

`ERROR at line X:` *`identifier_name`* `is undeclared.`

Rule 2 (Multiple Declarations): *An identifier can be declared only once.*

In the declarations block, identifiers to be used in the program are introduced. An identifier can be introduced only once, and it has to be an input, or an output, or a node. Therefore, having an identifier being declared multiple times is not allowed, even when these declarations are of different types (input, output, node). For example, if an identifier is declared as an input and also as a node, this is still an error. If an error occurs, the following error message will be printed:

`ERROR at line X:` *`identifier_name`* `is declared as a(n)` *`identifier_type`*`.`

Rule 3 (Unused Inputs and Nodes): *Every input and node must be used.*

All input and node identifiers introduced in the declarations block must be used somewhere in the circuit design block. In other words, if an identifier is declared as an input or a node, then this identifier has to appear within the expression on the right-hand side of some assignment given in the circuit design block. The important part here is the line number that should be printed if an error occurs the line number of the declaration of the identifier, and the following error message will be printed:

`ERROR at line X:` *`identifier_name`* `is not used.`

Rule 4 (Unassigned Nodes and Outputs): *Every identifier that is defined in the declarations block as a node or as an output must be assigned.*

If an identifier is declared as a "node" or as an "output" in the declarations block, it has to be assigned in the circuit design block. In other words, it has to appear on the left-hand side of an assignment in the circuit design block. The important part here is the line number that should be printed if an error occurs is the line number of the declaration of the identifier, and the following error message will be printed:

`ERROR at line X:` *`identifier_name`* `is not assigned.`

Rule 5 (Multiple Assignment to Nodes and Outputs): *An identifier that is defined in the declarations block as a node or as an output cannot be assigned multiple times.*

For an identifier that is declared as a "node" or as an "output" in the declarations block, multiple assignment statements cannot be assigned to this identifier. Therefore, we can have only one assignment statement where we see this identifier on the left-hand side. If an error occurs, the following error message will be printed:

`ERROR at line X:` *`identifier_name`* `is already assigned.`

Rule 6 (Unassigned Inputs): *Every input identifier has to be assigned in each circuit evaluation statement.*

If an identifier is declared as an "input" in the declarations block, then in each circuit evaluation, this identifier must be assigned to a value (which will be a constant boolean value, as guaranteed by the grammar). The important part here is the line number that should be printed if an error occurs; the line number of the `evaluate` keyword appears, and the following error message will be printed:

```
ERROR at line X: identifier_name is not assigned.
```

Rule 7 (Multiple Assignment to Inputs): *An input cannot be assigned multiple times in an evaluation.*

If an identifier is declared as an "input" in the declarations block, then there cannot be multiple assignments to this identifier in a circuit evaluation statement. The important part here is the line number that should be printed if an error occurs; the line number of the identifier appears, and the following error message will be printed:

```
ERROR at line X: identifier_name is already assigned.
```

Rule 8 (Incorrect Assignments to Inputs): *An input can only be assigned in the circuit evaluations.*

If an identifier is declared as an "input" in the declarations block, it cannot be assigned in the circuit design block. If such an error occurs; following error message willthe line number of the identifier appears to be printed:

```
ERROR at line X: identifier_name is already assigned.
```

Rule 9 (Incorrect Assignments to Nodes and Outputs): *A node or an output can only be assigned in the circuit design block.*

If an identifier is declared as a "node" or an "output" in the declarations block, it cannot be assigned in a circuit evaluation statement. The important part here is the line number that should be printed if an error occurs; the line number of the identifier appears, and the following error message will be printed:

```
ERROR at line X: identifier_name is not an input.
```

Errors will be printed in the order they appear in the program. That means the errors with the earliest line number will be printed first. If two or more errors occur on the same line, they will also be printed in the order they are seen (left to right). An example program is below:

```
1   input input1, input2
2   node node1, node2, input1, node2
3   output output1, output2
4
```

```
5     node1 = input3 or not input4
6     node2 = node1 and input2
7     output1 = input1 and node2
8     output2 = node4 or node2
9     output3 = node3 xor input2 and not input3
10
11    evaluate circuit1 (input1 = true, input2 = true)
12    evaluate circuit2 (input2 = false, input3 = false)
13    evaluate circuit3 (input2 = true, input1 = false)
14    evaluate circuit4 (input2 = false, input4 = true)
```

the following should be printed out:

```
ERROR at line 2: input1 is already declared as an input.
ERROR at line 2: node2 is already declared as a node.
ERROR at line 5: input3 is undeclared.
ERROR at line 5: input4 is undeclared.
ERROR at line 8: node4 is undeclared.
ERROR at line 9: output3 is undeclared.
ERROR at line 9: node3 is undeclared.
ERROR at line 9: input3 is undeclared.
ERROR at line 12: input3 is undeclared.
ERROR at line 12: input1 is not assigned.
ERROR at line 14: input4 is undeclared.
ERROR at line 14: input1 is not assigned.
```

For another example program:

```
1     input input1, input2, input3
2     node node1, node2, node3
3
4     input input3, input4
5     output output1, output2, output3
6     node node4
7
8     node1 = input1 or not input2
9     node2 = node1 and input2
10    output1 = input1 and node2
11    output2 = node1 or node2 or node4
12    output2 = node1 xor input2 and not output1
13
14    evaluate circuit1 (input1 = true, input2 = true)
15    evaluate circuit2 (input2 = false,
16                       input2 = false, input3 = false)
17    evaluate circuit2 (input2 = false, input1 = false, input3 = true)
```

should be printed out the following error messages.

```
ERROR at line 1: input3 is not used.
ERROR at line 2: node3 is not used.
ERROR at line 2: node3 is not assigned.
ERROR at line 4: input3 is already declared as an input.
ERROR at line 4: input4 is not used.
ERROR at line 5: output3 is not assigned.
ERROR at line 6: node4 is not assigned.
ERROR at line 12: output2 is already assigned.
ERROR at line 14: input3 is not assigned.
ERROR at line 14: input4 is not assigned.
ERROR at line 15: input1 is not assigned.
ERROR at line 15: input4 is not assigned.
ERROR at line 16: input2 is already assigned.
ERROR at line 17: input4 is not assigned.
```

4 Circuit Evaluations Results

After your semantic analyzer performs the semantic checks mentioned in Section 3, and if no errors are found in the end, it will move on to the second phase, which is evaluating the results of the circuit evaluation statements. In other words, it computes and prints the values of the outputs, if the inputs take the values given in the evaluate statements. This analysis is performed for each circuit evaluation statement separately.

Please keep in mind that the results will only be evaluated if the program contains no errors.

Note that, we are basically designing a logic circuit by an LCD program. Input identifiers are the inputs to be applied to the circuit. Output identifiers are the outputs to be produced by the circuit. Assignments give the connections of the logic gates in this circuit. For the circuit evaluation results, you need to compute the logic value that will appear at the outputs, when the values given in a circuit evaluation statement are applied to the inputs. For the program below:

```
input input1, input2
node node1, node2
output output1, output2

node1 = input1 or not node2
node2 = not input2
output1 = input1 and node2
output2 = node1 or node2

evaluate myCircuit1 (input1 = true, input2 = true)
evaluate myCircuit2 (input2 = false, input1 = true)
```

the output should be the following:

```
myCircuit1:output1=false,output2=true
myCircuit2:output1=true,output2=true
```

Note that, there are no spaces used in the output printed out, including the end of the line. That is, **do not insert any space** characters at the end of the lines you print. The outputs must be ordered in the order of their appearance in the declarations block.

5 Example Programs and Outputs

1. If the program is not grammatically correct then like the second homework you have to print ***ERROR***. For example, if we have the program below:

```
input A, Y
output K
evaluate myCirc(A = true, Y = false)
```

Then the output should be:

```
ERROR
```

2. If the program is empty, your code should not complain about it and should not print anything. (Similar to the expected outcome of the second homework. But this time there is no need to print ***OK***.)

```



```

Then there should be **no output** for this program.

3. If a program is grammatically correct but contains violations of the semantic rules then the output should display all the semantic errors. For example, if we have the following program:

```
input CS305, IS
node AN
output AWESOME, CLASS

AN = CS305 or not IS
AWESOME = PROGRAMMING and AN

evaluate LANGUAGES (CS305 = false, CS305 = true)
evaluate LCD (CS305 = true)

```

Then the output of the above program must be as follows:

```
ERROR at line 3: CLASS is not assigned.
ERROR at line 6: PROGRAMMING is undeclared.
ERROR at line 8: CS305 is already assigned.
ERROR at line 8: IS is not assigned.
ERROR at line 9: IS is not assigned.
```

4. If a program is grammatically correct and does not contain any violations of the semantic rules then the output should display the notifications. For example, if we have the following program:

```
input input1, input2
node node1, node2
output output1, output2

node1 = node2 xor not input2
node2 = not input1
output1 = input2 and node1
output2 = node1 or not node2

evaluate circuit1 (input1 = true, input2 = true)
evaluate circuit2 (input2 = true, input1 = false)

```

Then the output for the above program must be as follows:

```
circuit1:output1=false,output2=true
circuit2:output1=true,output2=true
```

6 How to Submit

Submit your bison file and flex file named as `username-hw3.y` and `username-hw3.flx` respectively; where `username` is your SU-Net username. You may use additional files, such as a header file. Please also upload those files.

We will compile your files by using the following commands:

```
flex username-hw3.flx
bison -d username-hw3.y
gcc -o username-hw3 lex.yy.c username-hw3.tab.c -lfl
```

So, make sure that these three commands are enough to produce the executable. If we assume that there is an LCD file named `test1.lcd`, we will try out your parser by using the following command line:

```
username-hw3 < test1.lcd
```

7 Notes

- **Important**: Name your files as you are told and **don't zip them.** [-10 points otherwise]
- **Important**: Make sure you include the right file in your scanner and make sure you can compile your parser using the commands given in Section 6. If we are not able to compile your code with those commands your **grade will be zero for this homework.**
- **Important**: Five test cases are shared on the SSH server. Meanwhile, the golden will be shared one week later than the assignment date for this homework.
- **Important: Since this homework is evaluated automatically make sure your output is exactly as it is supposed to be. Some of the points that we can think of are:**
  - **There should be no extra space at the beginning or at the end of a line.**
  - **Make sure that the spelling is as it is given in the homework document.**
  - **We check in a case-sensitive manner (e.g. ”ERROR” $\neq$ ”error”)**
  - **If you are not sure about your outputs you can compare your outputs with the outputs given by the golden.**
- You may get help from our TA or from your friends. However, **you must implement the homework by yourself**.
- Start working on the homework immediately.
- If you develop your code or create your test files on your own computer (not on cs305.sabanciuniv.edu), there can be incompatibilities once you transfer them to the cs305 machine. Since the grading will be done automatically on the cs305 machine, we strongly encourage you to do your development on the cs305 machine, or at least test your code on the cs305 machine before submitting it. If you prefer not to test your implementation on the cs305 machine, this means you accept to take the risks of incompatibilities. Even if you may have spent hours on the homework, you can easily get 0 due to such incompatibilities.

LATE SUBMISSION POLICY

Late submission is allowed subject to the following conditions:

- Your homework grade will be decided by multiplying what you get from the test cases by a “submission time factor (STF)”.
- If you submit on time (i.e. before the deadline), your STF is 1. So, you don’t lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse’s timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.